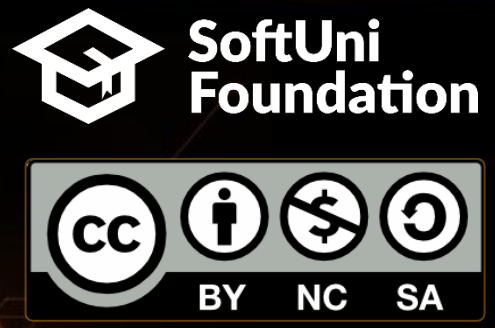
Enumerations and Annotations

Organizing Code



SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Contents

Questions

SoftUni Foundation

sli.do

#JavaFundamentals

SoftUni Foundation

Enumerations

Sets of Constant Objects

Enumerations

- Object type, limited to a set of values

```
enum Season {
   SPRING, SUMMER, AUTUMN, WINTER;
}
```

```
public static void main() {
   Season summer = Season.SUMMER;
   // use enum…
}
```

Enumerations in switch-case

- You can use enum values in switch-case statements

```
Season season = Season.SUMMER;
switch (season) {
   case SPRING: //… break;
   case SUMMER: //… break;
   case AUTUMN: //… break;
   case WINTER: //… break;
}
```

Enum Values

- `values()` – returns an array with all constants

```
Season[] seasons = Season.values();


for (Season season : seasons) {
  System.out.println(season);
}
```

default `toString()` returns the name

Enum Values (2)

SoftUni Foundation

- Every **enum** constant has a **zero based** order value

```
enum Season {
   SPRING, SUMMER, AUTUMN, WINTER;
}
```

```
Season season = Season.AUTUMN;
System.out.println(season.ordinal());
System.out.println(Season.AUTUMN.ordinal());
```

AUTUMN has index 2

Enum Names

SoftUni Foundation

- `name()` – gets the string representation

```
Season season = Season.AUTUMN;


System.out.println(season.name());
System.out.println(Season.WINTER);
```

`toString()` returns the name

- `toString()` can be overridden
- `name()` is `final`

Overriding toString()

SoftUni Foundation

- Overriding the method is considered to be a **good practice**

```
enum Season {
   SPRING, SUMMER, AUTUMN, WINTER;

   @Override
   public String toString() {
      return super.name().toLowerCase();
   } }
```

Enum Access

SoftUni Foundation

- `valueOf(Class, String)` – gets the `enum` by a given class and a `String`

```
Season spring =
    Season.valueOf(Season.class, "SPRING");


Season summer =
    Enum.valueOf(Season.class, "SUMMER");
```

`Enum` class

Problem: Weekdays

SoftUni Foundation

- Create Enum **Weekday with days** from Monday through Sunday
  - `toString()` should return weekdays in format "Monday"
- Class `WeeklyCalendar`
  - `void addEntry(String weekday, String notes)`
  - `Iterable<WeeklyEntry> getWeeklySchedule()`
- Class `WeeklyEntry(String weekday, String notes)`
  - `toString()` – "{weekday} - {notes}", ex. "Monday - sport"

Sorted by weekday

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/524#0

Solution: Weekday

SoftUni Foundation

```
public enum Weekday {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
    SATURDAY,
    SUNDAY;

    @Override
    public String toString() {
        String lower = super.name().substring(1).toLowerCase();
        return super.name().charAt(0) + lower;
    }
}
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/524#0

Solution: Weekday (2)

SoftUni Foundation

```
public class WeeklyEntry {
  public final static Comparator<WeeklyEntry> BY_WEEKDAY = getCompByDay();
  private Weekday weekday;  private String notes;
  public WeeklyEntry(String weekday, String notes) {
    this.weekday = Enum.valueOf(Weekday.class, weekday.toUpperCase());
    this.notes = notes;
  }
  @Override public String toString() { … }
  private static Comparator<WeeklyEntry> getCompByDay() {
    return (e1, e2) -> e1.weekday.compareTo(e2.weekday);
  } }
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/524#0

Solution: Weekday (3)

SoftUni Foundation

```
public class WeeklyCalendar {
  private List<WeeklyEntry> entries;
  public WeeklyCalendar() { this.entries = new ArrayList<>(); }
  public void addEntry(String weekday, String notes) {
    this.entries.add(new WeeklyEntry(weekday, notes));
  }
  public Iterable<WeeklyEntry> getWeeklySchedule() {
    Collections.sort(entries, WeeklyEntry.BY_WEEKDAY);
    return this.entries;
  }
}
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/524#0

Comparing Enums

- All **enums** are **Comparable**

```
Season spring = Season.SPRING;
Season winter = Season.WINTER;
if (spring.compareTo(winter) < 0) {
  System.out.println("%s comes before %s",
    spring, winter);
}
```

Comparison of ordinal values

Enum Methods

SoftUni Foundation

- Enums in Java can have **fields**, **methods** and **constructors**

```
enum Season {
  SPRING(28), SUMMER(44), AUTUMN(22), WINTER(4);
  private int max;
  Season(int maxTemperature) {
    this.max = maxTemperature;
  }
  public int getMaxTemp() { return this.max; }
}
```

Problem: Warning Levels

- Create a classes **`Logger`** and **`Message`**
- Create enum **`Importance`** - Low, Normal, Medium, High
- Record all messages **above given importance level**
- Logger should have **`Iterable<Message> getMessages()`**
- Create I/O client in Main

Record only HIGH and above

```
HIGH
NORMAL: All systems running
HIGH: Leakage in core room
LOW: Food delivery
END
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/524#0

Solution: Warning Levels

SoftUni Foundation

```
public enum Importance {
    LOW, NORMAL, MEDIUM, HIGH
}
```

```
public class Message {
  // private fields
  public Message(Importance level, String content) {
    this.level = level;
    this.content = content;
  }
  public Importance getLevel() { return level; }
  @Override public String toString() { … }}
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/524#0

Solution: Warning Levels (2)

SoftUni Foundation

```
public class Logger {
  // private fields…
  public Logger(Importance level) {
    this.level = level;
    this.messages = new ArrayList<>();
  }
  public void log(Message message) {
    if (message.getLevel().compareTo(this.level) >= 0) {
      this.messages.add(message); } }
  public Iterable<Message> getMessages() { return this.messages; }
}
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/524#0

Problem: Coffee Machine

- Class `CoffeeMachine`
  - `void buyCoffee(String size, String type)`
  - `void insertCoin(String coin)`
  - `Iterable<Coffee> coffeesSold()`
- Enum `CoffeeType` - Espresso, Latte, Irish
- Enum `Coin` – 1, 2, 5, 10, 20, 50 (ONE, TWO, etc.)
- Enum `CoffeeSize` – Small (50 ml, 50 c), Normal (100 ml, 75 c), Double (200 ml, 100 c)

```
TEN
TWENTY
TWENTY
Small Espresso
END
```

Single coffee sold
- "Small Espresso"

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/524#0

Solution: Coffee Machine


```
public enum Coin {
  ONE(1), TWO(2), FIVE(5), TEN(10), TWENTY(20), FIFTY(50);
  private int c;
  Coin(int c) {
    this.c = c;
  }
  public int getValue() {
    return c;
  }
}
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/524#0

Solution: Coffee Machine (2)

SoftUni Foundation

```
public enum CoffeeSize {
  SMALL(50, 50), NORMAL(100, 75), DOUBLE(200, 100);
  private int ml;
  private int c;
  CoffeeSize(int ml, int c) {
    this.ml = ml;
    this.c = c;
  }
  public int getMl() { return ml; }
  public int getPrice() { return c; }
  @Override public String toString() { … }  }
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/524#0

Solution: Coffee Machine (3)

SoftUni Foundation

```
public class CoffeeMachine {
  //TODO: Add constructor, fields, other methods…

  public void buyCoffee(String size, String type) {
    CoffeeSize coffeeSize =
CoffeeSize.valueOf(size.toUpperCase());
    CoffeeType coffeeType =
CoffeeType.valueOf(type.toUpperCase());
    Coffee coffee = new Coffee(coffeeSize, coffeeType);
                              //Continues on next slide
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/524#0

Solution: Coffee Machine (4)

SoftUni Foundation

```
// …
int price = coffee.getPrice();
    int currentSum =
this.coins.stream().mapToInt(Coin::getValue).sum();
    if (currentSum > price) {
      this.coffeeList.add(coffee);
      this.coins.clear();
    }
  }
}
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/524#0

SoftUni Foundation

Working with Enumerations

Live Exercises in Class (Lab)

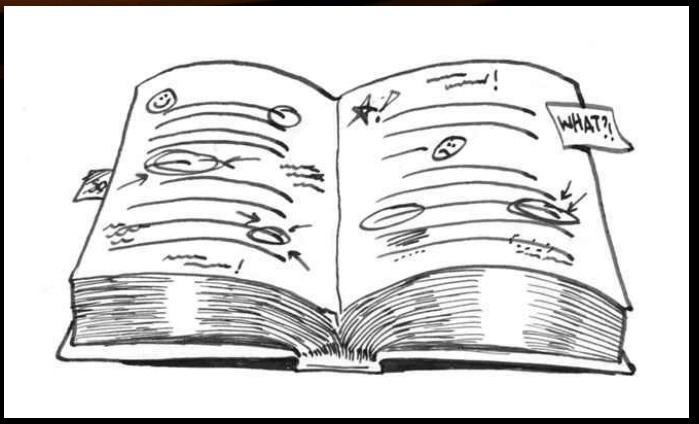
SoftUni Foundation

Annotations

Data about the data

Annotation

- **Data holding** class
- **Describes** parts of your code
- Applied to: **Classes**, **Fields**, **Methods**, etc.




```
@Deprecated
public void deprecatedMethod {
  System.out.println("Deprecated!");
}
```

Annotation Usage

SoftUni Foundation

- To generate **compiler messages** or **errors**

```
@SuppressWarnings("unchecked")
@Deprecated
```

- As tools
  - **Code generation** tools
  - **Documentation generation** tools
  - **Testing** Frameworks
- At runtime – **ORM**, **Serialization** etc.

Built-in Annotations

SoftUni Foundation

- **@Override** – generates **compile time error** if the method does not override a method in a parent class

```
@Override
public String toString() {
  return "new toString() method";
}
```

Built-in Annotations (2)

- `@SupressWarning` – turns off **compiler warnings**

```
@SuppressWarnings(value = "unchecked")
public <T> void warning(int size) {
  T[] unchecked = (T[]) new Object[size];
}
```

Annotation with value

Generates compiler warning

Built-in Annotations (3)

SoftUni Foundation

- **@Deprecated** – generates a **compiler warning** if the element is used

Generates compiler warning

```
@Deprecated
public void deprecatedMethod {
  System.out.println("Deprecated!");
}
```

Creating Annotations

SoftUni Foundation

- `@interface` – the keyword for annotations

```
public @interface MyAnnotation {
   String myValue() default "default";
}
```

Annotation element

```
@MyAnnotation(myValue = "value")
public void annotatedMethod {
   System.out.println("I am annotated");
}
```

Skip name if you have only one value named "value"

Annotation Elements

SoftUni Foundation

- Allowed types for annotation elements:
  - Primitive types (`int`, `long`, `boolean`, etc.)
  - `String`
  - `Class`
  - `Enum`
  - `Annotation`
  - `Arrays` of any of the above

Meta Annotations – @Target

SoftUni Foundation

- **Meta annotations** annotate annotations
- **@Target** – specifies where the annotation is applicable

```
@Target(ElementType.FIELD)
public @interface FieldAnnotation {
}
```

Used to annotate fields only

- Available element types – **CONSTRUCTOR, FIELD, LOCAL_VARIABLE, METHOD, PACKAGE, PARAMETER, TYPE**

Meta Annotations – @Retention

SoftUni Foundation

- `@Retention` – specifies where annotation is available

```
@Retention(RetentionPolicy.RUNTIME)
public @interface RuntimeAnnotation {
   // …
}
```

You can get info at runtime

- Available retention policies – `SOURCE, CLASS, RUNTIME`

Problem: Create Annotation

SoftUni Foundation

- Create annotation **Subject** with a String[] **element** "categories"
  - Should be **available at runtime**
  - Can be **placed** only **on types**

```
@Subject(categories = {"Test", "Annotations"})
public class TestClass {

}
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/524#0

Solution: Create Annotation

SoftUni Foundation

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Subject {
    String[] categories();
}
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/524#0

Accessing Annotation

SoftUni Foundation

- Some annotations can be accessed **at runtime**

```
@Author(name = "Gosho")
public class AuthoredClass {
  public static void main(String[] args) {
    Class cl = AuthoredClass.class;
    Author author =
       (Author) cl.getAnnotation(Author.class);
    System.out.println(author.name());
  }}
```

Accessing Annotation (2)

SoftUni Foundation

- Some annotations can be accessed **at runtime**

```
Class cl = AuthoredClass.class;
Annotation[] annotations = cl.getAnnotations();
for (Annotation annotation : annotations) {
  if (annotation instanceof Author) {
    Author author = (Author) annotation;
    System.out.println(author.name());
  }
}
```

Problem: Coding Tracker

SoftUni Foundation

- Create annotation **Author** with a String **element** "name"
  - Should be **available at runtime**
  - Should be **placed** only **on methods**
- Create a class **`Tracker`** with a method:
  - **`static void printMethodsByAuthor()`**

```
@Author(name = "Pesho")
public static void printMethodsByAuthor(Class<?> cl) {...}

@Author(name = "Gosho")
public static void main(String[] args) {
    Tracker.printMethodsByAuthor(Tracker.class);
}
```

```
"C:\Program Files\Java\jdk1.8.0_91\bin\java" ...
Pesho: printMethodsByAuthor()
Gosho: main()
```

Solution: Coding Tracker

SoftUni Foundation

```
public class Tracker {
  static void printMethodsByAuthor(Class<?> cl){
    Map<String, List<String>> methodsByAuthor = new
HashMap<>();
    Method[] methods = cl.getDeclaredMethods();

    for (Method method : methods) {
      Author annotation = method.getAnnotation(Author.class);
                                        //Continues on next slide
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/524#0

Solution: Coding Tracker(2)

SoftUni Foundation

```
if (annotation != null) {
    methodsByAuthor
        .putIfAbsent(annotation.name(), new ArrayList<>());
    methodsByAuthor
        .get(annotation.name()).add(method.getName() + "()");
    }
  }
  // TODO: print the results
} }
```

Check your solution here: https://judge.softuni.bg/Contests/Compete/Index/524#0

Summary

- Enumerations are **constant classes**
- Use enumerations to **organize code**
- Enumerations can have **fields**, **constructors** and **methods**
- **Annotations** are used to **describe** our code
- Annotations **provide** the **possibility to work** with **non-existing classes**
- Some annotations **can be accessed through reflection**

Enumerations and Annotations

SoftUni Foundation

Questions?

https://softuni.bg/courses/

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

Software University

SoftUni Foundation

FORUM

License

SoftUni Foundation

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license

- Attribution: this work may contain portions from
  - "Fundamentals of Computer Programming with Java" book by Svetlin Nakov & Co. under CC-BY-SA license
  - "C# Part I" course by Telerik Academy under CC-BY-NC-SA license
  - "C# Part II" course by Telerik Academy under CC-BY-NC-SA license